AR-010-406

Capturing and Analysing Usage of Interactive Computer-Based Tools

M.P. Phillips and R.J. Vernik

DSTO-RR-0119

APPROVED FOR PUBLIC RELEASE

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

Capturing and Analysing Usage of Interactive Computer-Based Tools

M.P.Phillips and R.J.Vernik

Information Technology Division
Electronics and Surveillance Research Laboratory

DSTO-RR-0119

ABSTRACT

This report argues that computer-based tools should incorporate features that support the capture and analysis of usage information, particularly if these tools are to be used as part of a research program. In addition to discussing the issues that need to be considered in terms of tool instrumentation and analysis support, this report provides details of experiences gained through the instrumentation of a Computer Aided Software Engineering (CASE) tool.

RELEASE LIMITATION

Approved for public release

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

Published by

DSTO Electronics and Surveillance Research Laboratory
PO Box 1500
Salisbury South Australia 5108

Telephone: (08) 8259 5555
Fax: (08) 8259 6567

AR No. AR-010-406
December 1997

APPROVED FOR PUBLIC RELEASE

Capturing and Analysing Usage of Interactive Computer-Based Tools

Executive Summary

Computer-based tools are generally developed to support individuals engaged in particular types of tasks. They also provide a basis for conducting research into Human Computer Interfacing (HCI) and information logistics (ie providing needed information to a user in an appropriate form at a particular point in time). Although many computer-based tools have been developed, relatively few have incorporated a means of capturing and analysing usage information. This report argues that tool instrumentation and analysis features need to be considered as part of the tool design, particularly if the tool is being developed as part of a research program. The information obtained could be used to assess the effectiveness of the tool for particular tasks and users, usage patterns, and user information requirements.

This report discusses a range of issues associated with the capture and analysis of tool usage. The report focuses on experiences gained in instrumenting a Computer Aided Software Engineering (CASE) tool that was developed as part of the Advanced Visualisation and Description of Software (AViDeS) program of research. An overview of the tool instrumentation approach is provided. The report also provides details of a Usage Monitoring and Analysis System (UMAS) which was developed to store, analyse, and present usage information captured by the instrumented tool.

The results of this work have shown that, in addition to the research value gained through tool instrumentation, several other benefits can result. For example, usage information was used to provide evidence that particular tasks had been carried out and for recording task outcomes. The information was also used as a basis for capturing, automating, and improving software engineering tasks.

Authors

Matthew Phillips
Information Technology Division

Matthew Phillips is a researcher employed in Software Engineering Group, Information Technology Division. His research interests include distributed systems, programming language design, software visualisation and object-oriented software engineering. Matthew has a Bachelor of Computer Science (with Honours) from The University of Adelaide.

Rudi Vernik
Information Technology Division

Rudi Vernik is a Senior Research Scientist employed in Software Engineering Group, Information Technology Division. He currently leads the Advanced Visualisation and Description of Software (AViDeS) task. His research focuses on software systems visualisation, large-scale software engineering, and information logistics. Rudi has a Bachelor of Electronic Engineering (with Distinction), a Diploma of Communications Engineering from the Royal Melbourne Institute of Technology and a PhD in Computer and Information Science from the University of South Australia.

Contents

List of Figures

List of Tables

Abbreviations

AViDeS	Advanced Visualisation and Description of Software
CASE	Computer Aided Software Engineering
FCS	Fire Control System
ITD	Information Technology Division
IV&V	Independent Verification and Validation
IVD	Integrated Visual Description
IVES	Integrated Visualisation Environment for Software
JORN	Jindalee Over-the-horizon Radar Network
PIR	Project Information Resource
PIRMAS	Project Information Resource Measurement and Analysis System
SWSC	Submarine Warfare Systems Centre
UMAS	Usage Monitor Analysis System

1. Introduction

1.1 Background

Although Computer-Aided Software Engineering (CASE) tools have promised to address many of the problems that have beset Software Engineering, there has been a marked lack of success in transitioning these tools into practice (Kemerer 1992) and (Vernik 1996 §2.2.2.3). This lack of success can be partly attributed to the fact that many CASE tools have not been effective for particular tasks (Budgen, Marashi et al. 1993). For example, a tool might effectively support software development but be unable to provide information in a form suitable for other processes (eg joint reviews). In addition, there has been a notable lack of integration between CASE tools and existing Software Engineering processes. Similar types of problems have also been encountered with computer-based tools in other domains (Jorgenson, Kritz et al. 1995).

We believe that another important reason why CASE tools have not completely fulfilled their promise is that there has been a lack of research into how these tools are used. The designers of computer-based tools need to carefully consider issues related to the provision and use of information: specifically, they need to know whether a tool is effective in providing the information required by users. This equates to knowing what tasks are being performed and what information is required for a given task at a particular point in time (Fernstrom 1991).

In this report we argue that research into computer-based tools needs to provide information on what tasks are performed, when tasks are performed, what tasks require which information categories, who performs particular tasks, and what tasks are effectively supported by particular tools. We believe that this information can be obtained in a reliable, non-intrusive way if the tools themselves are instrumented so that usage data can be automatically captured as they are used.

As discussed in Section Summary of Experiences, our experiences suggest that in addition to its application within research into tool use and information provision, usage data has a number of other important applications:

- Usage data provides a basis for process automation and facilitation of common tasks.
- Usage data provides an 'audit trail' of what information has been accessed by whom and when. Thus usage data can be useful as part of project documentation because it records that particular tasks (eg software inspections) have been effectively completed.
- Usage capture acts as a feedback mechanism from the user to tool developers on the tool's usefulness, limitations and problems.

These applications of usage data are discussed in §5.

The research into capture and analysis of usage data described in this report has been conducted as part of the Advanced Visualisation and Description of Software (AViDeS) task being carried out by the Software Engineering Group, ITD, DSTO. The AViDeS task was established to address problems of visibility and information use on large software projects

and the outcomes have been new concepts and tools for software visualisation. One such tool is the SEE-Ada environment (Appendix A), which implements a model-based system visualisation approach (Vernik 1996), providing computer-based visualisations that allow customisation and adaptation of software project information to user needs. SEE-Ada has been used as a key apparatus for the AViDeS research and, as such, has been instrumented to capture usage data in line with research objectives. This report discusses our experiences in capturing and analysing SEE-Ada usage data.

1.2 Purpose

This report aims to communicate experiences and results of instrumenting a CASE tool to support investigations into how the tool is used and its effectiveness in facilitating information access. It describes the instrumentation approach and the tools developed to help analyse captured usage data. The report also gives a brief summary of experiences in deploying the tool in case studies as well as presenting some preliminary conclusions about the effectiveness of such instrumentation and the advantages of instrumenting CASE and other computer-based tools in general.

Although this report contains experimental data and preliminary conclusions, it should be noted that this data has been collected primarily from a single industry case study. It is not our intention in this report to draw final conclusions from the data presented, but rather to demonstrate the need for usage monitoring and to describe the general principles and techniques we have developed in order to collect and analyse usage data.

1.3 Presentation

Section 2 provides a brief overview of the AViDeS objectives, concepts, and approaches. It describes key AViDeS concepts including the process-based framework and the integrated visualisation approach. Section 2 also introduces the concept of an Integrated Visualisation Environment for Software (IVES) and discusses how an IVES implementation (SEE-Ada Version 3) has been used in laboratory and field studies. These studies have helped verify the AViDeS concepts and have aided in gaining a better understanding of how computer-based visualisation techniques can be used to support the information needs of individuals engaged in software engineering tasks.

Section 3 discusses the principles that we believe determine what information should be captured in a computer-based tool and lists the data captured from SEE-Ada that satisfies these information requirements. Following this, the approach taken to instrument SEE-Ada to support usage monitoring is outlined. The usage monitoring subsystem added to SEE-Ada—the SEE-Ada Usage Monitor—is then discussed.

Section 4 describes the Usage Monitor Analysis System (UMAS), which is the tool used to store and analyse captured usage information. This section demonstrates how UMAS is used to filter, analyse and display usage data.

Section 5 presents experiences and outcomes from using the SEE-Ada Usage Monitor in conjunction with UMAS on a medium-sized software project. A scheme developed for classifying tasks within a software engineering project is described, along with some

conclusions about the general applicability of such a scheme. The possibility of synthesising task classifications based on task content is then discussed. Some usage overview data collected over the course of the project is also presented. The important role of the Usage Monitor in recording, enacting and improving software engineering processes and enhancing project documentation is highlighted. We also present recommendations for CASE tool design based on our experiences with SEE-Ada.

Section 6 discusses the need for further work in collecting usage data and developing a new, more widely applicable follow-on to SEE-Ada. The need for further research into developing a task classification system based on usage profiles is also discussed.

Section 7 provides conclusions drawn from our research into tool instrumentation. It is argued that instrumentation is indispensable for CASE tool research and development. We also highlight that usage information is a valuable addition to the metrics and documentation of a software development project.

Appendix A provides an overview of the SEE-Ada software system visualisation tool and some further references. Appendix B contains the complete list of task definitions described in Section 5.1. Appendix C contains a version of the schema used by UMAS to store usage information. Appendix D lists the usage profiles taken from the FCS project discussed in Section 5.3.

2. AViDeS Concepts and Approaches

The experiences discussed in this report are based on work undertaken as part of the Advanced Visualisation and Description of Software (AViDeS) task (ALO 94/081). This section provides the necessary context for this report by providing background information on the AViDeS research including fundamental concepts and approaches.

2.1 Overview of AViDeS Research

The purpose of the AViDeS task is to address problems being experienced by Defence in its acquisition, development and support of large military software systems. These include: problems in evaluating and maintaining software due to poor product and project visibility; and, problems with the effectiveness of, and high costs associated with, many current forms of project and product information (eg. documentation, metrics).

The AViDeS work focuses on defining techniques which provide enhanced software product and project visibility through more effective use of underlying project information (eg. as captured by integrated project support environments and tools). It is based on presenting information by the way of computer-based visualisation techniques rather than through the production of vast amounts of software paperwork and metrics as is current practice for many large software projects.

2.2 Key Concepts

2.2.1 Process-Based Framework

The Contextual Model of Figure 2-1 helps focus attention on those areas of prime interest to the AViDeS research. These are the processes and tasks undertaken by individuals involved in the software engineering aspects of a project, the source of information available, and the information flows. The Software Engineering (SE) Process includes those standard software lifecycle processes (as defined in the international standard on software lifecycle processes (ISO/IEC_12207-1995 1995)) that are required to generate and deliver software products as specified by the customer's requirement. The lifecycle processes provide an overview of the types of SE functions that are performed as part of a software project. Individuals undertake particular tasks within this framework. For example, the technical management process might include tasks that monitor project status, inspection tasks that identify quality problems, and analysis tasks that support problem resolution. The AViDeS research is primarily interested in how information can be most effectively provided to support individuals engaged in particular tasks.

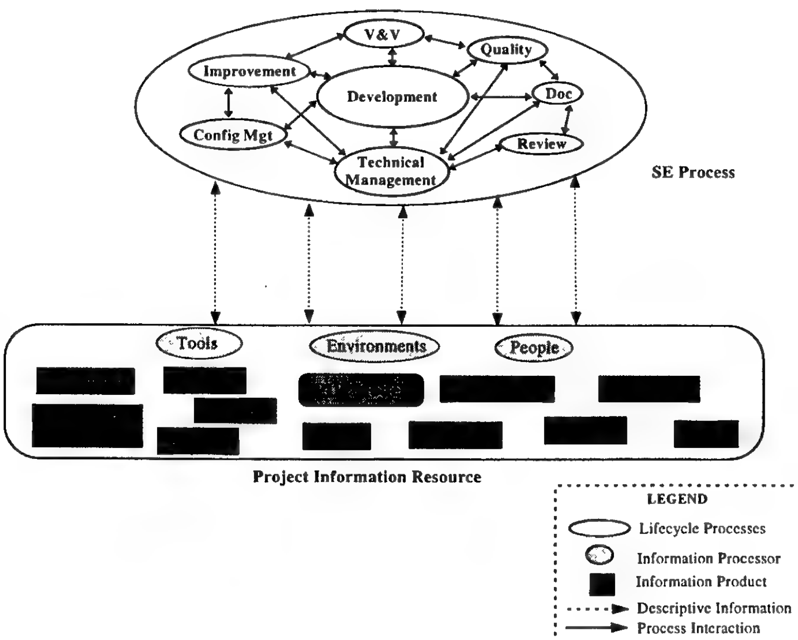


Figure 2-1. Contextual model

The Contextual Model also focuses on the source of information available to those engaged in SE tasks. This source of project information is defined as the Project Information

Resource (PIR). The PIR is a conceptual entity that represents the totality of persistent information for a project. (Vernik 1996) provides an approach to modelling and measuring the PIR. This work has been used to gain insights into the diversity, quantity, and timing of information produced during a software project.

The various lifecycle processes produce and use information. Information can be used directly (eg by referencing a document) or it can be accessed through interactions with tools, environments, or people. A wide range of problems and issues related to the provision and use of software project information have been reported (see Vernik 1996 §2.3 for an overview). These include problems of information overload, inaccessibility, availability, and cost. A particular problem relates to the assimilation and integration of information. For example, some tasks may require a user to assimilate information from a range of sources and of different forms. A user tasked with evaluating the maintainability of a software product might need to assimilate test information, architecture diagrams, and configuration information. Some tasks require core information to be provided together with contextual information. For example, metric information often needs to be used together with other information to establish the relationship between specific metrics and the attributes of an entity that the metric values are meant to describe. The assimilation of information of different forms can require significant cognitive effort. Approaches are needed to help reduce this cognitive load by integrating information and providing it in an appropriate composite form.

The Description Process was defined in Vernik (1996) to help address the many problems and issues related to the provision and use of information. As shown in Figure 2-2, this process provides a basis for accessing and filtering of information from the PIR. This process also supports the customisation, tailoring and adaptation of information. Four sub-processes were defined as part of an overall process framework. These are the Provision, Use, Mediation, and Facilitation processes. The Provision Process provides the basis for accessing and filtering of underlying information. It also supports the transformation of this information into composite Description Products. The Use Process considers the way in which provided information is used to support particular tasks, taking into account human cognitive and action processes (Norman 1986). The Facilitation Process supports the provision of custom descriptions. Facilitation is based on an analysis of information needs for particular tasks. The process supports the generation of description specifications which are used by the Provision Process to provide information customised to user needs. The Mediation Process provides a mechanism for direct feedback and sharing of knowledge between the Provision and Use processes. This process allows information to be adapted to user needs.

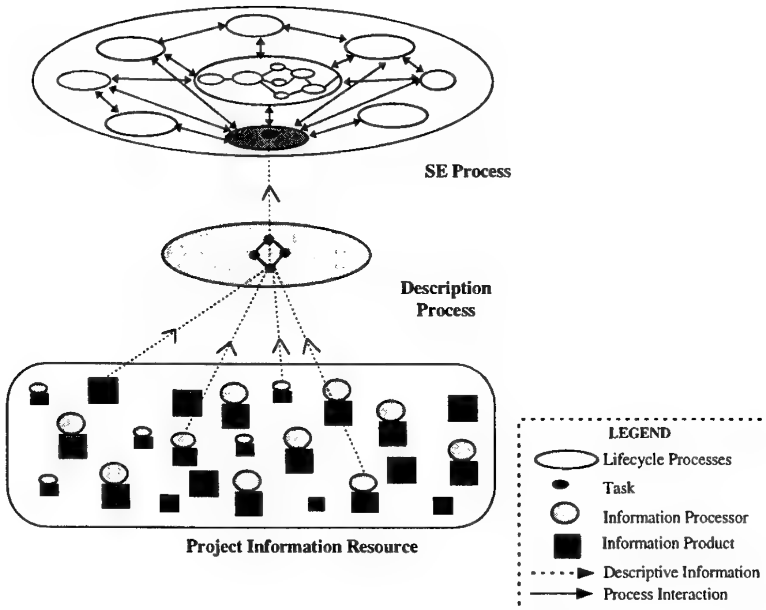


Figure 2-2. Including a description process

2.2.2 Integrated Visualisation Approach

The AViDeS research focuses on the use of computer-based visualisation approaches as a basis for providing information tailored to a particular task and user. An integrated visualisation approach (Vernik 1996) has been proposed as a means of supporting the Description Process. This approach is based on the use of a Composite Systems Model to support the integration and assimilation of information. This model also supports the integration of representations and views.

The concept of an Integrated Visualisation Environment for Software (IVES) has been proposed as a means of applying the integrated visualisation approach to the software engineering domain. Figure 2-3 shows how an IVES supports the Description Process and how it functions within a software project context. The IVES provides support for accessing, filtering, and integrating PIR information by way of a Software Product Model (SPM). The SPM is the Composite Systems Model for this domain of interest. It includes those entities, relationships, and attributes that are used to describe the Software Product. For example, the SPM could comprise the structural entities of a product (eg design and code entities) and the relationships between them (eg encapsulation, inheritance, uses, calls etc.).

A composite SPM is then generated by adding attribute values. For example, attributes such as size, complexity, test results, configuration status, and requirements satisfied could be defined for the structural entities.

The SPM provides the basis for generating Integrated Visual Descriptions (IVDs) for the user. IVDs are classes of information products that use a set of computer-based visualisations to describe aspects of interest. They provide a flexible means of customising and adapting information to suit individual needs.

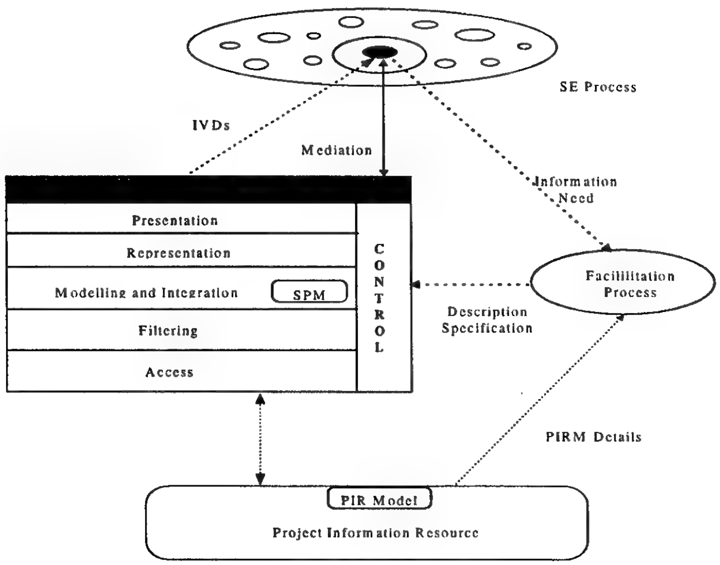


Figure 2-3. IVES conceptual model

The IVES can be controlled directly by the user through a direct manipulation user interface (Hutchins, Hollan et al. 1986) hence supporting the mediation process. The environment can also be controlled by way of a description specification which allows custom descriptions to be prepared as part of a Facilitation Process. The control component of an IVES provides a means of controlling the functions performed by a particular layer or by sequencing actions of several layers.

2.3 IVES Implementation

The SEE-Ada Version 3 environment has been developed as a practical implementation of an IVES. Appendix A provides an overview of this environment. SEE-Ada has been used to support AViDeS research objectives by providing a vehicle for developing and testing concepts and approaches.

Figure 2-4 shows how SEE-Ada has been used within a software project context. The PIR represents the totality of information available to those involved in software-related tasks. In this case, the project relies heavily on information captured by an integrated software engineering environment, the Rational Apex Environment (Apex 1993). Several other sources of information are also shown including the Rational Rose analysis and design tool,

various documentation (both deliverable documents such as those produced under (DOD-STD-2167A 1987) and the developmental information captured in Software Development Files), product metrics information, test information, and information that resides with project personnel. SEE-Ada has a range of filters and mechanisms for capturing elements of this information and generating a composite SPM. For example, logical design information captured from the Rose tool is used as part of the structural model. The model is extended with physical design and code structure information obtained from the Apex design facility and code libraries. When combined with traceability relationships, the SPM represents a consistent structural model of the software product ranging from the highest level design concepts through to particular lines of code. A composite SPM is generated by integrating attribute information. This information is filtered from particular sources and includes test results, size measures, change and configuration details, build status, and information on requirements traceability. The composite SPM can be automatically generated and updated by way of the Facilitation Process. This process uses information captured by the PIR Modelling and Analysis System (PIRMAS). PIRMAS can provide information on the various classes of information that are available at particular points in time and provides details on how this information can be accessed and filtered.

The user can obtain required information by directly interacting with the environment to produce the required visualisations. Alternatively, by using the Script Mode interface, the Facilitation Process can be used to automatically generate custom descriptions based on user needs and the usage context. Typically, a combination of both is required where a custom description is generated and the user interacts (or mediates) with the environment to adapt it to their exact needs.

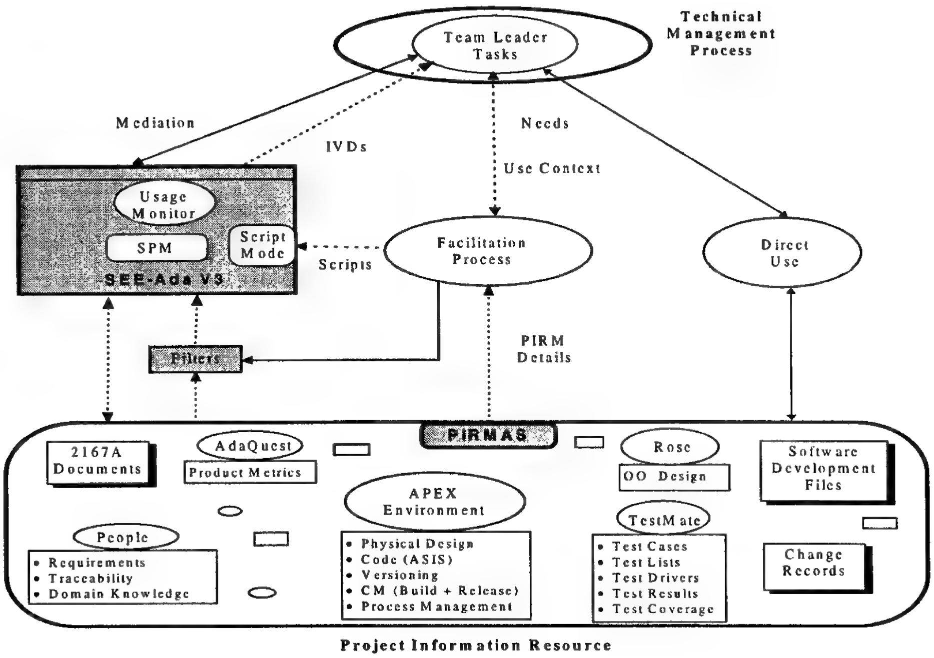


Figure 2-4. Implementing the IVES concepts in a real project setting

2.4 Experimentation

The setup shown in Figure 2-4 has been used as the basis for both laboratory and industry studies (Vernik 1996). One of these projects was the Active Missile Decoy (AMD) Fire Control System (FCS). The primary focus of these studies related to the information needs of Software Team Leaders. These studies aimed to identify the types of tasks performed by Team Leaders, what information was required to support these tasks, and how the AViDeS concepts and approaches could be used to more effectively provide this information.

Clearly, to conduct these types of investigations and to verify the AViDeS concepts, some form of instrumentation was required. A Usage Monitor was incorporated into SEE-Ada to help support the aims of the studies. The next section discusses the approach taken, and issues associated with providing a means of capturing and analysing usage information.

3. Instrumentation Principles and Approach

This section discusses aspects that need to be considered in relation to tool instrumentation. The approach taken to enable usage monitoring within SEE-Ada is then described.

3.1 Instrumentation Principles

Instrumentation goals need to be clearly defined before deciding what information must be captured. For example, one goal may be to capture what information is being accessed. Another goal might be to measure the effectiveness of data representations provided by the tool. Once goals have been defined, a set of generic information characteristics needed for each goal can be identified. The next step is to produce a mapping from these generic characteristics to specific information items that can be captured from the tool. This type of goal-based approach is described by (Basili and Rombach 1988) and extended by (Vernik 1996).

Our main goals for SEE-Ada instrumentation are related to the identification of user information requirements and measurement of tool effectiveness. From these goals we identified that the following characteristics needed to be captured for each task executed:

1. **Task definition and objectives**. The task should be defined using a classification scheme that is effective for task analysis (a task classification scheme that was developed during case studies is described in §5.1).

2. **User details**. Knowing who performs what tasks—in conjunction with knowledge of what role each user has—makes it possible to know the information needs for a given class of user.

3. **Information accessed**. This includes what entities, relationships and attributes were displayed. It is important that this characteristic be related to the representation used to display the information (characteristic 4).

4. **Representations used**. This captures what sort of information displays were used and how they were applied. Display types might include directed graphs, text lists, tables, hierarchy trees, etc. This characteristic should also capture what facilitation processes were used to generate each display.

5. **Date/time of recording and elapsed time information**. Time-stamping each task allows profiles of task types executed over time to be generated. It also allows us to see when different information types are accessed. The elapsed time for each task is useful as an effectiveness metric for facilitation processes.

6. **The degree of user interaction**. This is an indication of how much effort the user had to expend to obtain the information required. This characteristic also provides another measure of facilitation improvement in addition to characteristic 5.

7. **User feedback both during and at the end of the task**. Allowing the user to enter comments during the task helps us understand what the user's intentions were. These comments are especially valuable since they appear in context—we can see what the user was doing at the time the comment was made. Comments can also be used to record task results and achievements. In addition, having the user rate the effectiveness of the tool at the completion of each task allows us to check that efforts to facilitate information access and otherwise improve the tool are successful from the user's point of view. Note that when collecting effectiveness data it is important to realise that there are many possible effectiveness criteria that can be used to rate a

task: these criteria might include *accessibility, customisability, scalability,* etc. A more exhaustive set of effectiveness criteria is listed in (Vernik 1996 Appendix B).

Table 3-1 lists each characteristic above against information items captured in SEE-Ada.

Table 3-1. Mapping task characteristics to SEE-Ada information

Characteristic	SEE-Ada Information Items
1. Task definition	**Task name**: selected from a defined task naming scheme. **Objective**: user's description of task objective.
2. User details	**User name**: the Unix login name of the person recording the task.
3. Information accessed	**Ada entities**: Ada packages, procedures, etc.. **Design entities**: design abstractions such as subsystems, classes, etc. **Product attributes**: measures of code size, requirements, complexity, etc. **Process attributes**: test results, completion status, audit status, etc. **Resource attributes**: unit authorship, effort, etc. **Relationships**: package dependencies, procedure call relations, design encapsulation, etc.
4. Representations used	**View types**: SEE-Ada view names such as *List, Subsystem, Layers, Source, Contains,* etc.
5. Start date and elapsed time	**Start date.** **End date.** **Actual time**: actual time spent recording not counting pauses.
6. User interaction	**Number of commands executed.** **Keystrokes.** **Mouse clicks.**
7. User feedback	**User comments.** **Evaluation criteria**: feedback on effectiveness criteria that is entered into a form at task completion. Criteria might include: understandability, completeness, flexibility, etc.

3.2 SEE-Ada Instrumentation Approach

Figure 3-1 shows how the SEE-Ada usage monitoring system fits into the overall system introduced in Figure 2-4. The SEE-Ada Usage Monitor saves usage data in log files, which are processed by a filter tool and imported into the Usage Monitor Analysis System (§4). The usage logs can also be used as part of the facilitation process (§5.4, 5.5) and can form part of the total Project Information Resource as project documentation.

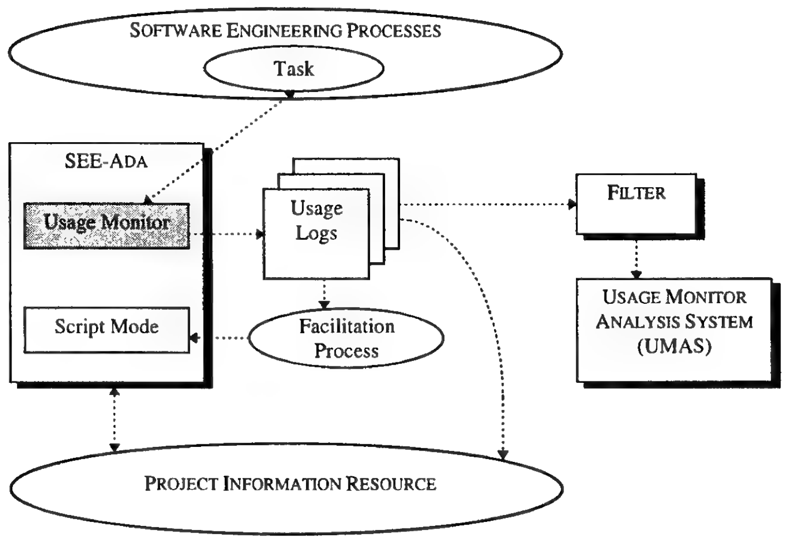


Figure 3-1. Usage monitoring system within the AViDeS process-based model

Two possible approaches to capturing the usage data described in Table 3-1 are possible:

- capture the data by having SEE-Ada fill in a table with fields similar to those listed in Table 3-1, or
- capture the data in log form, with a header containing the task definition, followed by a list of commands executed during the task and terminated by a footer containing data such as task execution time.

We chose to capture the usage data in a log form because not only does the log contain all the data described in Table 3-1 (accessible by analysis) it can also be used for other purposes including project documentation and process improvement. These uses are discussed in §5.4-5.6.

3.2.1 SEE-Ada Usage Monitor

The SEE-Ada Usage Monitor windows are shown in Figure 3-2. The Usage Monitor is a special mode of SEE-Ada and may only be enabled by a command line option when SEE-Ada is invoked. When the Usage Monitor is activated in this way, the SEE-Ada Main Window initially appears as an icon and the user sees a notice informing them that the Usage Monitor is active. This notice also contains an option to display an information window that describes how the user should employ the Usage Monitor for specific research investigations: this description is usually customised for each project.

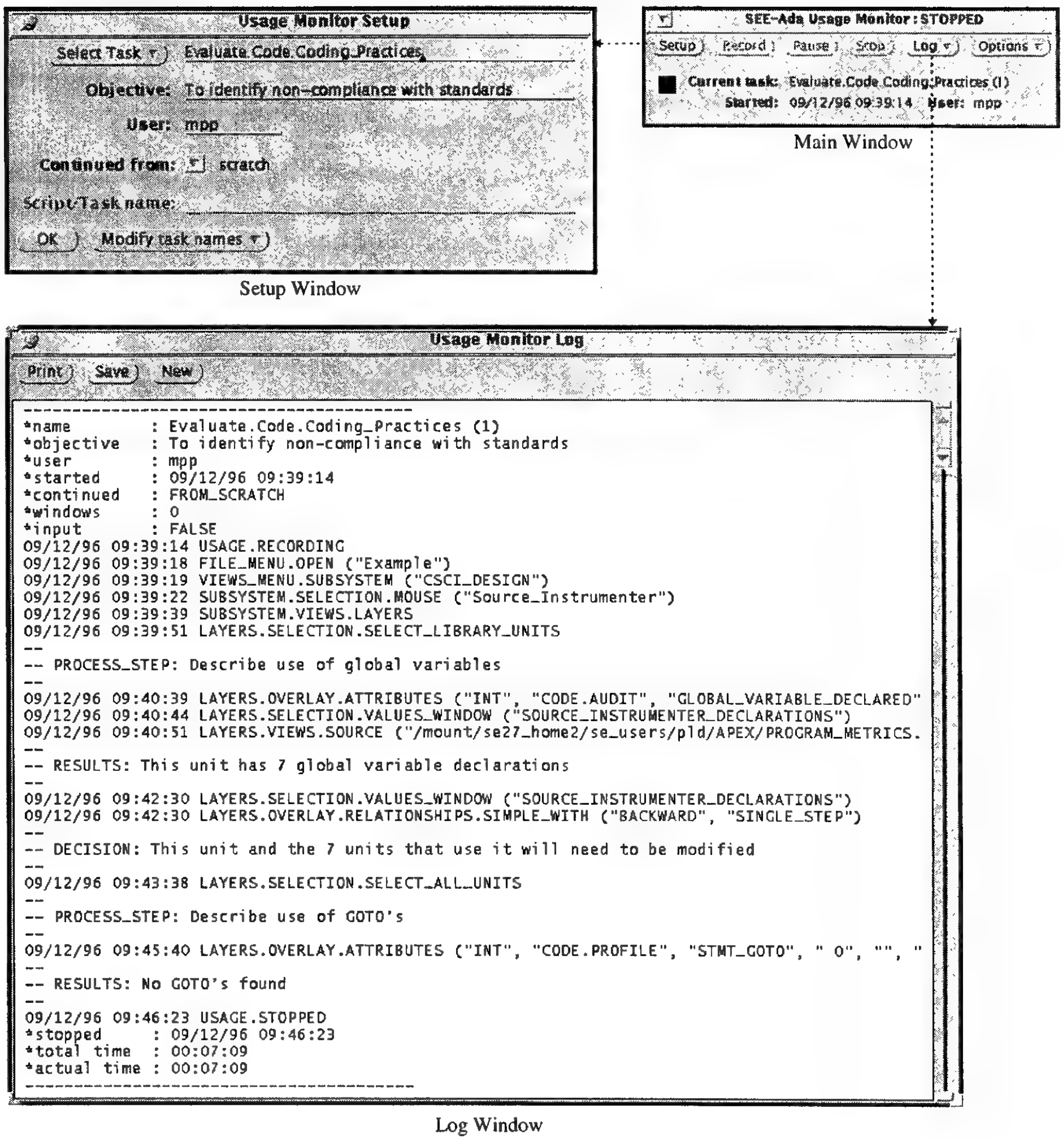


Figure 3-2. SEE-Ada usage monitor windows

After the initial notice, the Usage Monitor Window appears in the top-right corner of the screen and the Setup Window appears in the centre. The Usage Monitor Window displays information about the current task and the mode of the Usage Monitor: *Stopped, Recording* or *Paused.*

Whenever a new task is to be started, the user enters task details via the Setup Window. In the example above, the user *mpp* intends to check compliance of source code with standard coding practices, so the task identifier *Evaluate.Code.Coding_Practices* is selected and a

sentence describing the objective entered. The user then starts the task by selecting the *OK* button and begins using SEE-Ada. The commands executed during the task are shown in the Log Window, which is displayed via a menu on the Main Window. When the task is complete, the user selects the *Stop* button on the Main Window, which terminates the task. At this point the time spent executing the task is automatically entered at the end of the log and the Usage Monitor is ready to start the next task.

The design of Usage Monitor GUI was carefully considered because it is important to encourage the user to employ the Usage Monitor consistently and correctly. In order to facilitate this, the user should be aware of the current task and Usage Monitor mode. To this end the Main Window is small, stays on top of all windows and provides clear information on current task and visual feedback on recording state. The Setup Window is designed to allow the user to quickly select the task type from a menu, enter the task objective and then start executing the task.

There is also an alarm feature that causes a prompt window to appear at regular intervals (usually every 15 minutes) which asks the user if the they are still recording the same task. This helps to prevent *run-on* tasks: ie tasks that occur when the user forgets to inform the Usage Monitor that they have started a new task and merges one task into the other.

3.2.2 Usage Logs

Figure 3-2 shows an example of a SEE-Ada usage log. The log consists of a task header (the initial starred lines), a series of commands and a footer (trailing starred lines). The header contains all the information entered into the Setup Window with the addition of:

- the time the task started,
- whether the task was started from 'scratch' or was a follow-on to a previous task,
- the number of SEE-Ada windows that were open when the task began, and
- whether the user made any input to SEE-Ada before beginning the task.

The items described in the last three points are captured in order to detect if the user might have had information already on the screen when the task was started. This usually occurs when the user has windows from a previous task still open. If the user did start the task with information already displayed then the usage log may not have captured all the information accessed. Thus this data helps to verify that the Usage Monitor is being used correctly and is capturing all relevant data.

The lines following the header represent commands executed by the user. These are recorded one command per line, each command being prefixed by the time it was issued. Commands are represented in the same syntax as SEE-Ada Script Mode commands, allowing straightforward translation of usage logs into scripts. In order to make commands easy to understand, the Script Mode command structure is modelled as closely as possible on the logical SEE-Ada GUI command interface eg *FILE_MENU.QUIT*, *VIEW.LAYERS*, etc. Following each command is an optional parameter list containing any extra information required by that command eg *FILE_MENU.OPEN ("A_System_Name")*.

The lines starting with '--' are comments which are used to document usage logs. When entering a comment, the user may first select a comment category such as *OBSERVATION, PROCESS_STEP, PROBLEM, DECISION, RESULTS*, etc. These comment categories help to structure comment information. For example, in the task being recorded in Figure 3-2, the user has delimited individual sections of the evaluation with the *PROCESS_STEP* category, and tagged any actions to be taken as a result of the evaluation using *DECISION*. This scheme makes logs easier to understand and helps in finding specific types of comment during later analysis.

The footer of the log contains the time the task was stopped, the total time that the task was recording and the actual time spent executing the task (ie the total time minus any time spent paused). Any data collected via the optional task evaluation form is also recorded in the footer.

3.2.3 SEE-Ada Instrumentation

SEE-Ada's command logging feature is implemented as an independent subsystem that is connected into the original code by a number of 'hooks' so as to minimise the chance of introducing errors into SEE-Ada. While this is not the best way to implement a usage monitor in general, it was necessary in SEE-Ada because it does not have fully separate GUI and command layers. In a tool that has detached GUI and command layers, instrumentation would be best achieved by having the command subsystem report all commands received to the usage monitor subsystem. The need for this sort of design is discussed further in §5.7.

It was recognised early on that, while capturing all SEE-Ada commands would allow more complete 'macro' recording, it would also result in capturing much more information than was needed to satisfy research goals. Our approach was to initially instrument SEE-Ada to capture only a core set of commands encompassing major operations and then to progressively add commands in response to requirements found during use in case studies. Many of the newer captured commands were added in order to provide more complete information on the state of tool before major operations. For example, in Figure 3-2 the first command following the second comment records that the *SOURCE_INSTRUMENTER_DECLARATIONS* unit was selected just before the second command (which displays relationships for selected units) was invoked. Without this selection command it would not be possible in later analysis to determine for which units the relationship was displayed.

A later addition to the Usage Monitor was an option to capture counts of keystrokes and mouse clicks and insert them in the log as special commands. This was provided as an extra aid to measuring the amount of user interaction with the tool.

4. The Usage Monitoring and Analysis System

The Usage Monitor Analysis System (UMAS) is a tool for storing, analysing and presenting usage information collected by the SEE-Ada Usage Monitor. This section describes how UMAS is structured and what tools it provides for usage analysis and visualisation.

4.1 UMAS Overview

Figure 4-1 shows the structure of UMAS. The data stored in UMAS is generated by a Perl (Wall and Schwartz 1991) script that reads usage logs and produces task data in a form that can be directly imported into the UMAS database. Data scanned from the usage logs includes the fields from the task header, information about actions performed during the task and information types accessed. A complete version of the data schema used by UMAS is shown in Appendix C.

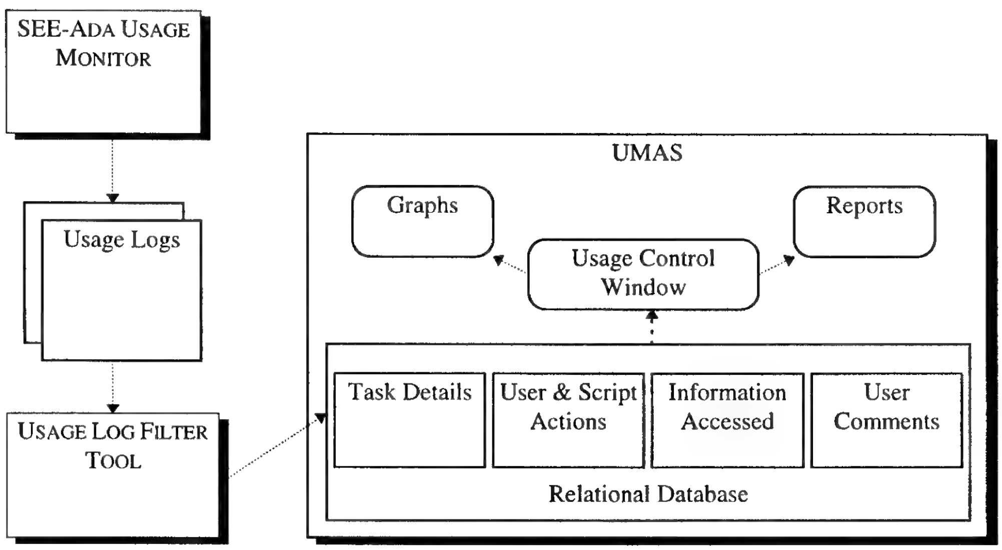


Figure 4-1. UMAS overview

UMAS provides queries, graphs and reports that allow access to usage data in a number of ways. The main interface is the Usage Control Window, which allows the user to filter and display tasks and task information.

4.2 Analysis and Display of Usage Information

The primary interface provided by UMAS to filter and display usage data is The Usage Control Window (Figure 4-2). This window is composed of a number of group boxes, each containing a one or more data fields. Each group box corresponds to an aspect of task data

stored with the database (eg task definition, views used, etc) while each of the fields within the group are filters against that aspect. The number in the top-right corner of each group box shows how many items currently match the filter, while the *View* button at the bottom of each group displays a list of those matching items.

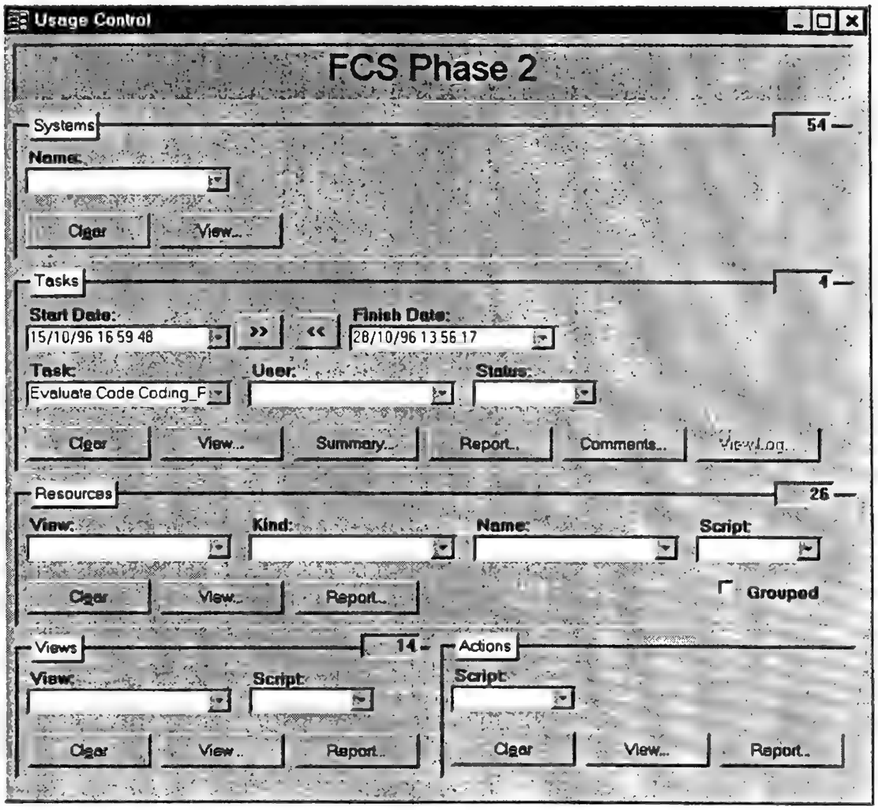


Figure 4-2. UMAS Usage Control window

The *Systems* and *Tasks* groups both serve to match a subset of tasks. While the *Systems* group is used to restrict the selected tasks to those that accessed particular SEE-Ada systems, the *Tasks* group restricts the task subset based on the fields in the task header (such as task definition, date recorded, user, etc). The other groups allow the user to filter subsets of detailed data associated with each task in the selected subset. This data includes the views used, scripts executed, and attributes applied. For example, the *Kind* field in the *Resources* group could be set to 'Attributes' so that only the attributes used by the task subset are displayed when the *View* button is selected.

Most UMAS reports take their data from the filtered subset generated by the Usage Control Window, thus the procedure for generating a particular report is to first select the data using the appropriate filters and then select the desired report type. For example, suppose a

summary of all *Evaluate.Code.Coding_Practices* tasks between *15/10/96* and *28/10/96* is required. The filters needed to select these tasks have been entered into the *Tasks* group in Figure 4-2. Once these filters have been entered, the summary report shown in Table 4-1 is displayed by selecting the *Summary* button.

Table 4-1. Filtered summary of UMAS tasks

Date	User	Time	Views	Attributes	Relations	User Actions	Script Actions	Comments	Prints	Scripts
16/10/96	cking	01:29:44	3	23		41	21	31	2	1
24/10/96	cking	00:11:49	3	2	1	22	2	1		1
28/10/96	cking	02:00:32	3			13		5	1	
28/10/96	cking	00:56:12	3			26		25	3	

UMAS also provides a number of graphs that display summary information over all tasks in the database. These graphs can help to answer questions such as 'who is recording what tasks?', 'what task types are most prevalent?' and 'what tasks are being executed at different stages of the project?'. Examples of these graphs in the context of the FCS project are presented in §5.2.

5. Summary of Experiences

This section contains some initial experiences gained from using the SEE-Ada Usage Monitor and UMAS on a medium-sized software development project. A more detailed discussion of these findings can be found in (Vernik 1996 §6.2.8.4).

As mentioned in §1.2, it should be noted that the data displayed in this report is taken from a single project that was halfway complete when the report was being produced. The data presented here is provided for illustrative purposes only, and any results derived from the data should not be regarded as final.

5.1 Capturing Task Definitions

Prior to commencing a usage study, task analysis needs to be performed. This analysis should result in a classification of tasks which will be used to define all work undertaken (one such classification of tasks is listed in Appendix B). This task classification is important because of the role it plays in structuring task analysis. Without a classification to use as the basis of referring to tasks it would not be possible to answer such questions as 'what work was undertaken?' or 'who is undertaking the work?'. The classification also helps the person executing the task to maintain a clear idea of what goal they are working towards.

Note, however that although the classification is important, it does not necessarily have to be definitive, and may only be applicable within a certain project. The important thing is that it is effective for analysis and applied consistently. Nevertheless, the task classification

will be more useful if it is generic, thus allowing direct comparison of results across multiple projects. To permit this, the classification must be able to evolve—within a consistent framework—as further studies are made. The framework used for the task classification was the international standard on software development processes (ISO/IEC_12207-1995 1995). The task classification scheme in Appendix B was initially developed from the processes defined in *ISO 12207* and was then extended by conducting interviews and making observations of tasks in the workplace.

One problem that arose while developing the task classification was that the same name is often used for quite different tasks. For example, the term *review* is often used for both code inspection tasks and formal meetings with the customer. Therefore, once an initial classification had been developed, it was important to ensure that users worked consistently within this framework and modified it only when appropriate. In practice this was not wholly successful: only a subset of the task definitions were used, sometimes in the wrong context, and little user-generated evolution of the task scheme eventuated. Another common problem occurred when the user started out on one task and then, after identifying an anomaly, began another task without resetting the Usage Monitor. Although measures have been put in place that should help resolve these problems, it may be more effective to classify tasks in terms of synthesised usage profiles (as suggested in §5.3 and (Vernik 1996 §7.3.1.2)).

One important conclusion we have drawn from our work in this area is that a task classification scheme works best when:

- the scheme is based on a well-defined framework,
- users are involved in defining the scheme,
- users are trained in the meaning and use of the scheme, and
- researchers are regularly present to supervise and provide advice.

5.2 Overview of Tool Usage

In addition to the reports available through the Usage Control Window, UMAS can generate a number of overview graphs and reports. These graphs display information including:

- what tasks are being executed,
- how long is being spent on each task type,
- who is executing what sort of tasks, and
- when different information types are being accessed.

For example, Figure 5-1 shows a cumulative total of time spent on each task category in the FCS project broken down into weekly intervals. Figure 5-2 is a snapshot of the last week (28/10) in Figure 5-1. The data displayed in these graphs is sampled from mid-way through the project, and it can be seen that the most time has been spent on the tasks *Analyse.Design.Features* (~17 hours) and *Analyse.Code.Dependencies* (~16 hours).

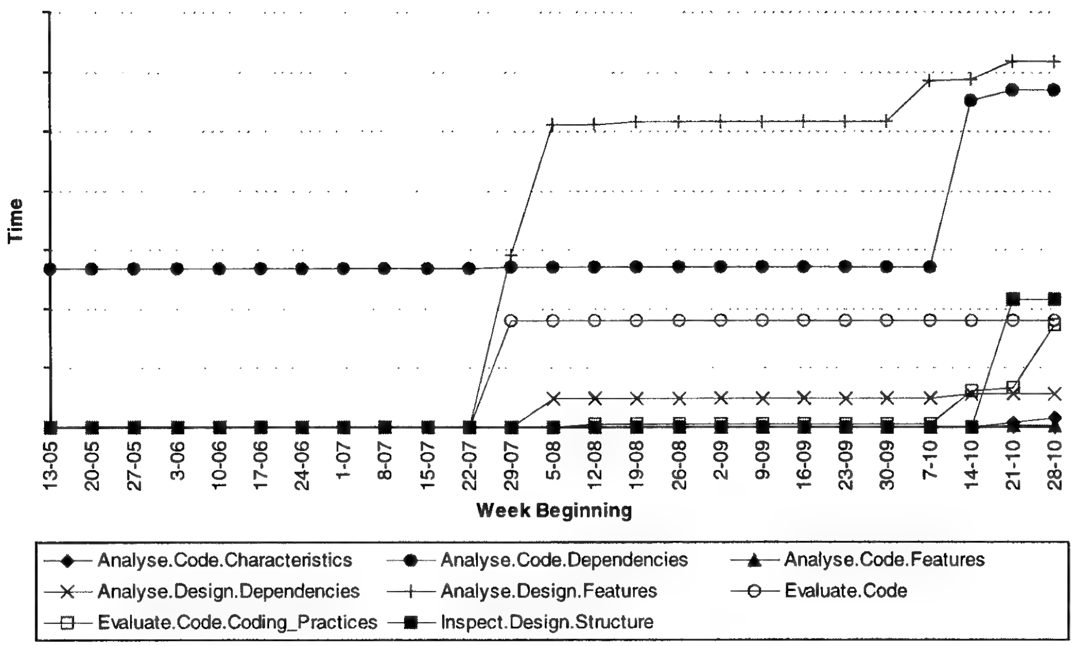


Figure 5-1. UMAS graph of time spent by task at weekly intervals

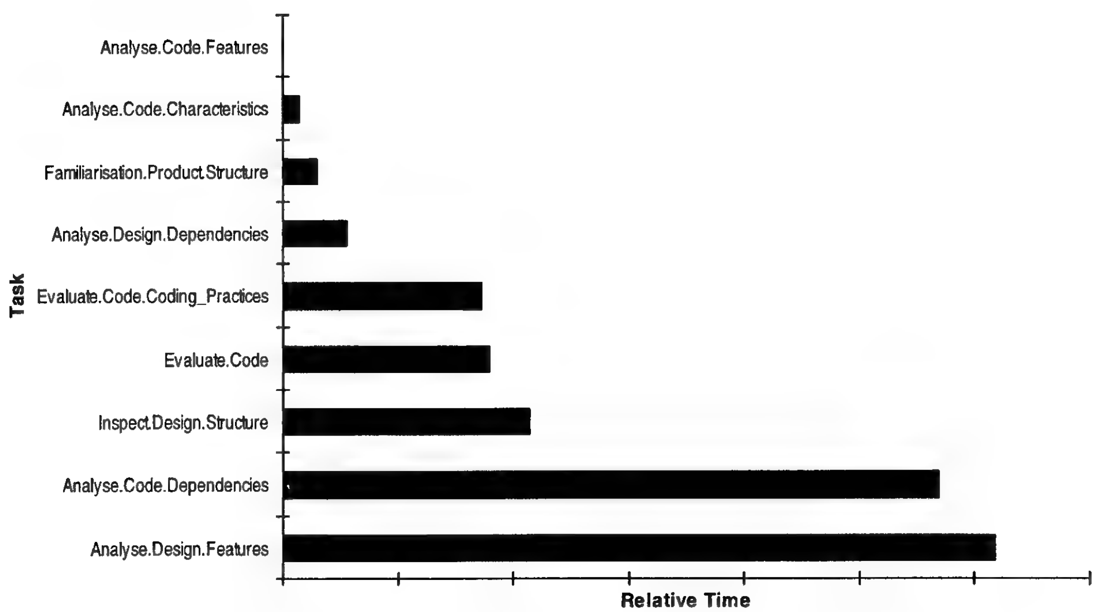


Figure 5-2. UMAS graph of time spent for each task category

These graphs provide objective answers to questions such as 'which tasks are executed at a particular stage in the project?' and 'when is this type of information accessed?' As a whole, UMAS graphs and reports form a project profile which is useful both to researchers and project managers because it provides objective measures of project activity and status at any given time. These displays can also serve as an education and training aid for task managers by showing when different tasks are usually executed and the sort of task profiles that might be expected at each stage of a project.

5.3 Usage Profiles and Task Classification

One form of UMAS report is the usage profile, examples of which are listed in Table 4-1 and Appendix D. The usage profile report contains a breakdown of various actions performed during a task, and was initially developed to show relative task activity levels. However, while using these reports on FCS project data, we observed that the profiles for identical task types are often similar. Consequently, if a profile differs significantly from others in the same task domain, it usually indicates that the associated task warrants further examination.

For example, Task 1 in Appendix D is a *Monitor* task in which the user is looking at Ada source code in order to assess the degree of completeness of various subsystems. The high number of user actions (41) is characteristic of this sort of task since it is exploratory in nature. In contrast, we would expect that tasks 14, 15 and 16—which are all *Evaluate.Code* tasks—would have a relatively low number of user actions, since this task is executed by a script. This expectation is borne out in tasks 14 and 16 which both have zero user actions, but not for task 15 which has 27. A study of the task 15 log reveals that the user saw a problem about halfway through the task and spent some time inspecting source code as a result. This source code inspection generated the high user action count because it involved manually selecting units and invoking a command to display their source code.

The fact that the profiles for the same task types are often similar and that large deviations from the profile hint at exceptional conditions indicates that profiles might be used as a task classification scheme. While the profile scheme illustrated here is too coarse to be generally useful as a classification mechanism, its (unintended) utility in this regard indicates that a task content-based classification scheme might be possible with a specifically designed profile method.

5.4 Process Recording and Enactment

As previously discussed, the usage log for a task can be used to generate a script that re-enacts that task. The Usage Monitor can also be used while a script is being executed, in which case it captures both user and script actions in the log.

Figure 5-3 shows the Usage Monitor and Script Mode working together in this manner. The Script Mode window shows a script that is being used to help automate the process of evaluating coding practices. The Usage Monitor has recorded both script and user actions as part of the current task with the script actions preceded by a '+'.

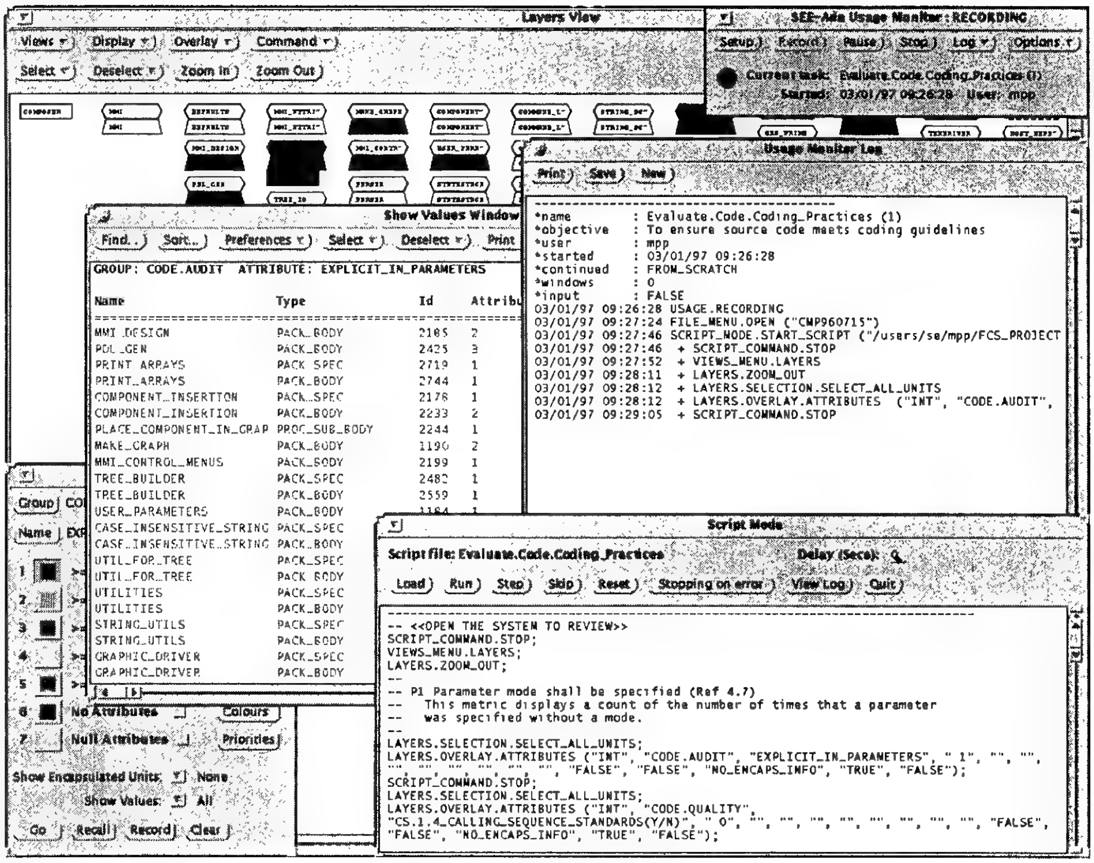


Figure 5-3. Usage Monitor and Script Mode working together

Using the Usage Monitor and Script Mode together has proved a very effective method of generating scripts for process enactment. For example, on the FCS project, the regular code evaluation task involves an experienced user following a process designed to discover potential and actual problems in the project source code. This task was facilitated by performing an analysis of its information requirements (based on project development standards) and then developing a mapping from those requirements to information and views available in SEE-Ada. The evaluation task was then performed by an experienced analyst using SEE-Ada to provide the information and views identified by the previous step, while the Usage Monitor was used to record the task. The usage log thus generated was then transformed into a script that would cause Script Mode to enact the evaluation within SEE-Ada.

5.5 Process and Tool Improvement

By recording the actions taken during a task, usage logs increase understanding of what is done during the task, what information is needed by the user and whether facilitation

processes are effective. This understanding allows the tools and facilitation processes to be improved.

For example, the Usage Monitor was used to enhance the task facilitation script described in §5.4. After the script had been developed, the usage logs generated when users executed the script were analysed in order to identify further enhancements. These enhancements included extra steps that users needed to execute and suggestions by users entered as comments in the log. This process was performed a number of times, producing a cycle of incremental process improvement.

The Usage Monitor also plays a role in conveying user feedback to researchers by allowing special comments in the log. UMAS can be used to find these comments and display the task logs where they occur. For example, if the tool behaves oddly or cannot provide some information that the user needs, a comment with the category *SEE_ADA_PROBLEM* can be logged at the point where the problem occurred. This feedback mechanism is both convenient for the user and effective in facilitating tool improvement. The number of problems logged by users can also serve as a metric of process improvement (Vernik 1996 §7.3.5).

5.6 Process Documentation

Although the Usage Monitor was developed to support research, when it was deployed in an industrial software development project it was also found to be useful as a documentation tool. Developers and task managers on the FCS project used Usage Monitor logs as documentation of which tasks had been completed, what units had been inspected, what problems were found, and task results. The comment feature of the Usage Monitor proved particularly useful in this regard, because it enabled observations to be recorded in the context that prompted them, thus allowing them to be brief while remaining readily understood at a later date.

5.7 Considerations for Computer-Based Tool Architectures

An important lesson learned during the course of adding the Usage Monitor to SEE-Ada is that it is important for such tools to have fully decoupled command and GUI layers. In other words, the tool should have two independent subsystems, one being a 'server' that executes commands sent to it by 'clients', the other being a GUI 'client' that provides a front end to the server and graphical display of server output.

Apart from the advantages from a software engineering point of view, there are two reasons to structure CASE tools this way:

- Instrumentation can be installed in the command layer to capture and filter all commands issued. This allows consistent and reliable command capture in a way that is unlikely to affect other functions of the tool. It also means that instrumentation can be added at any point in tool development with minimal impact on existing functionality.

- The input to the command layer can come from sources other than the GUI, making it straightforward to implement a scripting facility, which would simply be another client of the command layer.

The ability to execute scripts is an obvious advantage in any application, and we believe that the added power of employing a usage monitor and scripting system as a mechanism for process improvement make it a important part of any computer-based tool.

6. Further Work

We have already mentioned that the SEE-Ada Usage Monitor and UMAS system have so far been used only within DSTO and on a single industry case study. In order to formulate more general conclusions and enhance techniques further, usage data from a number of different projects needs to be collected and analysed. Accordingly, one of the main areas of future work is to deploy SEE-Ada on further case studies (eg the IV&V of software in the Collins Submarine, JORN and the Joint Command Support Environment).

As part of the follow-on to AViDeS research, an extended, more generic successor to SEE-Ada is being planned. The design of this new tool will incorporate the separation of command and GUI layers discussed in §5.7 in order to more completely support the facilitation process.

There is also a need to pursue research into the possibility of classifying tasks using the content-based profiles approach discussed in §5.3.

7. Conclusions

The research described in this report highlights the importance of instrumentation in CASE tools. In particular, it has been argued that usage information:

- provides knowledge of task-related information needs,
- facilitates incremental process and tool improvement, and
- can and should be used as part of software engineering documentation.

This report has discussed the principles of CASE tool instrumentation, including what characteristics should be captured and how these might map to data available in a CASE tool. The necessity of developing a task classification scheme has been discussed and we have described some difficulties associated with using such a scheme in the field. We have also suggested that classifying tasks using content-based profiles might be a possibility.

As a real-world example of CASE tool instrumentation, this report has described the approach taken to add a usage monitor to SEE-Ada. Experiences while undertaking this work have lead us to recommend that computer-based tools be designed from inception to include support for usage monitoring and scripting.

This report has also described UMAS, a tool developed for the analysis of SEE-Ada usage information. We have demonstrated the features developed in UMAS for filtering and displaying usage data. Example data collected from a case study conducted on the FCS project has shown how usage information can be used to profile software engineering tasks and information use.

References

Apex (1993). Using Rational Apex. Santa Clara, CA, Rational Software Corporation.

Basili, V. R. and H. D. Rombach (1988). "The TAME Project: Towards Improvement-Orientated Software Environments." IEEE Transactions on Software Engineering **14**(6): 758-773.

Budgen, D., M. Marashi, et al. (1993). Case Tools: Masters or Servants? Software Engineering Environments '93, Reading, England, IEEE.

DOD-STD-2167A (1987). Defence System Software Development, U.S. Department of Defence.

Fernstrom, C. (1991). The Eureka Software Factory: Concepts and Accomplishments. Proc. Third European Software Eng. Conf., Springer Verlag, Berlin.

Hutchins, E. L., J. D. Hollan, et al. (1986). Direct Manipulation Interfaces. User Centered System Design. D. A. Norman and S. W. Draper. Hillsdale NJ, Lawrence Erlbaum Associates: 31-61.

ISO/IEC_12207-1995 (1995). Information Technology - Software Life Cycle Processes, International Standards Organisation.

Jorgenson, L., R. Kritz, et al. (1995). Is Visualization Struggling under the Myth of Objectivity? Visualization '95, Atlanta, Georgia, IEEE Computer Society Press.

Kemerer, C. F. (1992). "How the Learning Curve Affects CASE Tool Adoption." IEEE Software **May 1992**: 23-28.

Norman, D. A. (1986). Cognitive Engineering. User Centered System Design. D. A. Norman and S. W. Draper. Hillsdale NJ, Lawrence Erlbaum Associates: 31-61.

Vernik, R. J. (1996). Visualisation and Description in Software Engineering. Computer and Information Science. Adelaide, University of South Australia: 232.

Wall, L. and R. L. Schwartz (1991). Programming Perl. Sebatopol U.S.A., O'Reilly & Associates.

Acknowledgments

The following people have supported the research described in this report:

Richard Altmann, Andrew Loja and Ian Turner helped develop the initial design for the SEE-Ada Script Mode interface. Ian Turner implemented Script Mode and has been responsible for enhancements needed to support this research. Chris King has provided invaluable feedback on the use SEE-Ada Usage Monitor in his role as Project Manager of the FCS project.

Appendix A - Overview of SEE-Ada

A.1 Introduction

SEE-Ada is a tool for the visualisation of large, complex Ada software systems. It uses computer graphics to provide meaningful, scalable views of the total software system, including design and code entities, their attributes and relationships. SEE-Ada can assist in a wide range of software engineering tasks including management, development, independent verification and validation, quality assessment, and software maintenance.

Key features of SEE-Ada are:

- Software System Visualisation environment based on the use of an underlying Software Product Model to support multiple-perspective views and information integration.
- Open environment that allows the import and integration of a wide range of information from a variety of project sources.
- Allows the display of project information integrated with structural representations of the software system.
- Provides the ability to customise and adapt information to specific needs.

A.2 System Framework

Figure A-1 shows the structure of the SEE-Ada environment.

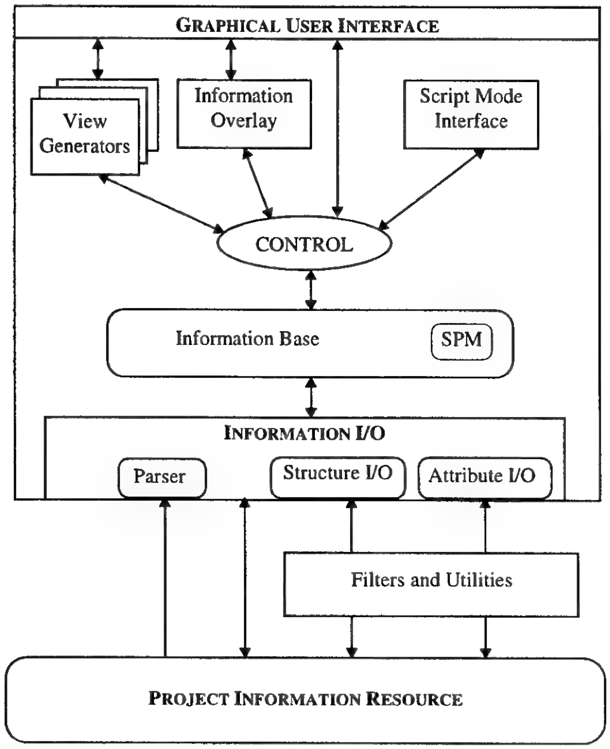


Figure A-1. The SEE-Ada System Framework

A range of information can be extracted from software project sources and imported into SEE-Ada. Structural information about the system (eg entities and their structural relationships) can be extracted from the Ada source code or obtained directly from an Ada compilation system via a standard ASIS interface using the filter tools provided. Structural design-level detail can also be imported by way of the Structure I/O feature. Data representing software/project attributes is imported into the SEE-Ada environment from external, commercial or locally developed tools. Many types of information can be imported including requirements, configuration management information, product measures, test results and so forth. Structural information provides the structural element of the Software Product Model (SPM). Other information is integrated into the model as attributes of the structural entities.

Architectural views of the software are generated from the structural model and displayed in graphical form. Attribute information can be integrated into the views and used to describe and provide insights into software characteristics.

SEE-Ada is an open system: design information, development history, and other data from CASE tools, development environments, and other sources can be imported into SEE-Ada and displayed in a consistent, and integrated manner.

A.3 SEE-Ada Views

Figure A-2 shows a set of integrated views of a software system as presented by the SEE-Ada product. These views are generated from information captured in the Software Product Model.

The Subsystem View shows 'design-level' information. In this case, the view shows the relationship between logical design entities (class categories filtered from the Rational Rose design tool) and physical design entities (Rational Subsystems from the Rational Apex environment).

The Layers View shows the Ada compilation units that implement the highlighted section of the design shown in the Subsystem View. These Ada units are arranged based on compilation dependencies to provide a compact, spatial representation of the code modules. This view can be customised and tailored to support user needs. The compact representation allows other information to be superimposed. For example, in Figure A-2, information on the degree of commenting is superimposed via a colour mapping mechanism. The entities shaded red show those source code modules that have no comments. Other colours have been used as threshold values to indicate the degree of commenting.

The Graph View is a view that can be generated from an arbitrary selection of entities. The directed graph representation can display any one of the relationships stored in the Software Product Model. The Graph View shown in Figure A-2 shows the *with* structure between a selection of packages.

The Contains View shows those subprograms encapsulated in an Ada compilation unit. For example, the TREE_BUILDER package body encapsulates both functions and procedures as shown in Figure A-2.

The user has selected the 'STRING_ASSIGN' procedure and requested a text view to show the related source code. Link attributes associated with the Software Product Model provide the basis for displaying this information.

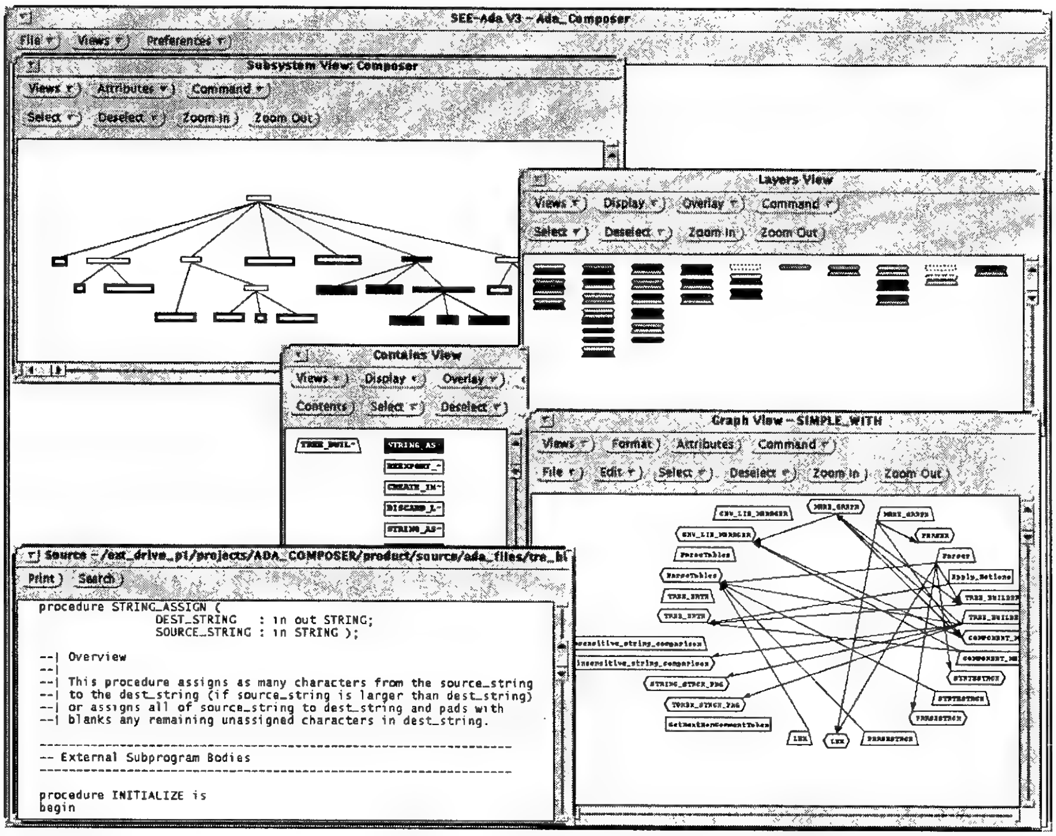


Figure A-2. Main Representations Used within SEE-Ada

As can be seen from this figure, a set of integrated views of the software product allows the user to quickly traverse from high-level design concepts to individual lines of code in a consistent way whilst maintaining context.

A.4 Viewing Attribute Information

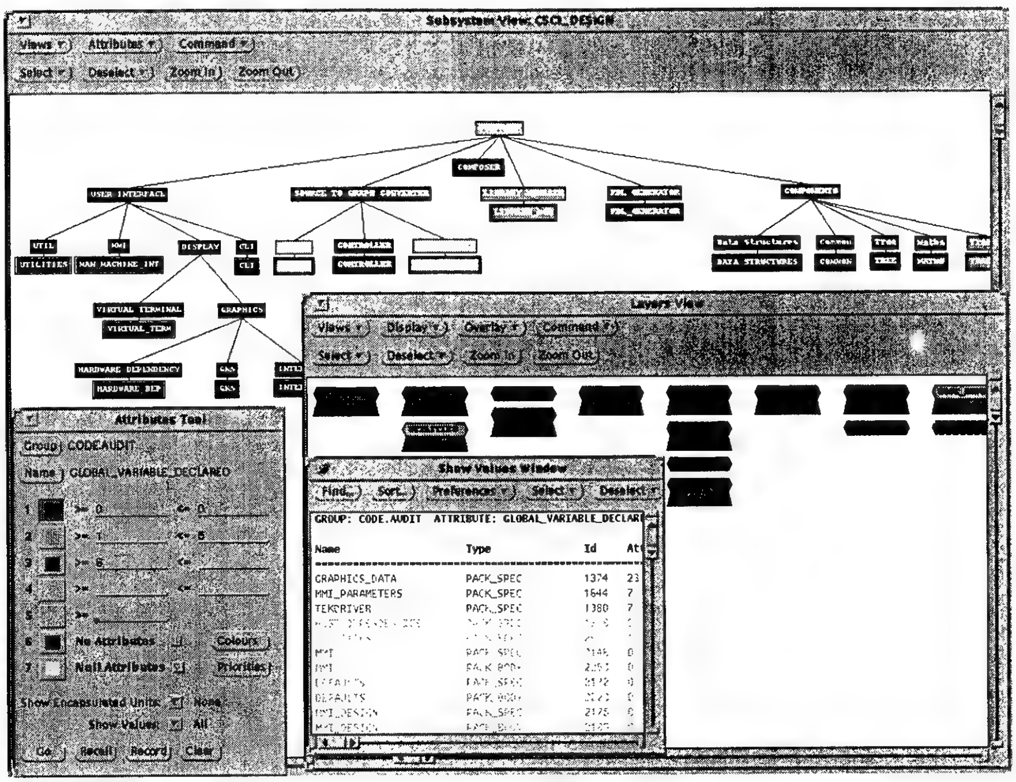


Figure A-3. Viewing Attributes in SEE-Ada

The integration of information is a key concept that has been explored as part of the AViDeS research. Figure A-3 shows a Subsystem View and a Layers View with attribute information superimposed.

In this case, attributes were used to identify those entities that declare global variables. The use of global variables can result in highly coupled software that is difficult to maintain. In Ada, the use of global variables in sections of the software which use concurrent threads can result in race conditions. These conditions can induce serious timing problems and intermittent failures that are difficult to rectify.

Attributes can be overlaid onto any of the Subsystem, Layers, Worksheet, Graph and Contains view via the use of the Attributes Tool as shown in Figure A-3. The Attributes Tool specifies the mapping of threshold values to up to 5 different colours. This allows both numeric and symbolic data to be overlaid as colours onto the 5 aforementioned views. Individual values can be viewed in a Show Values window if desired.

The approach of integrating information on SEE-Ada views can support a wide range of needs. For example, configuration management information can be used to identify those units that have undergone most change. This information can also be superimposed to show which programmers had authored or changed particular units. Test information can be superimposed to show which units have undergone test, the extent of that testing and the results of particular tests.

A.5 Viewing Relationships

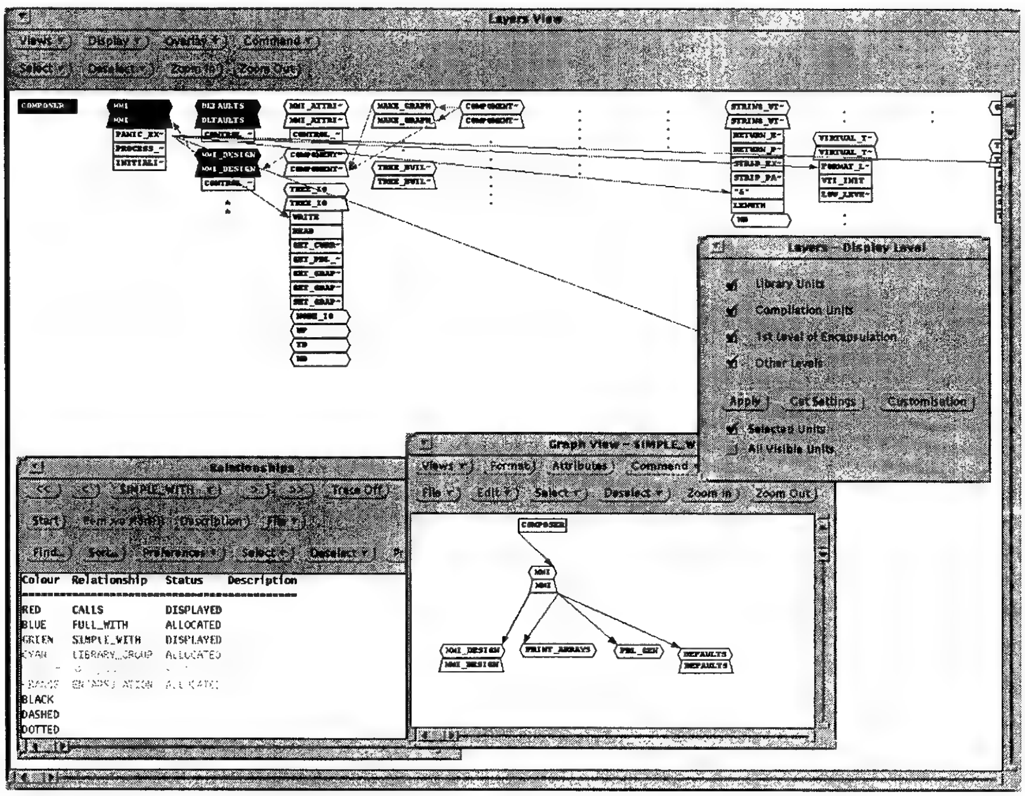


Figure A-4. Viewing Relationships in SEE-Ada

Figure A-4 shows how SEE-Ada views can be customised to provide required information. Relationships between code entities are typically provided in terms of a directed graph of the complete system (eg the Graph View shows the compilation dependency 'with' relationship between a subset of units selected in the Layers View). This approach does not scale well and the superfluous information often confuses the user.

The integrated visualisation approach as used in SEE-Ada allows the user to query for and superimpose only that information which is necessary for the task at hand. For example,

the first level of a call tree from a subprogram has been superimposed in 'Red' and the 'Green' trace line shows usage of the 'COMPONENT_MANAGER' library. Any relationship can be loaded into SEE-Ada. An example of the types of relationships that may be loaded is shown in the Relationships window of Figure A-4.

A benefit of the integrated visualisation approach is that the information can be customised and adapted for a particular need and the information can be presented in terms of a familiar context (ie the general shape and layout of the pre-arranged compilation unit lattice). The display detail level of individual compilation units can be set so as to remove clutter caused by irrelevant information. In Figure A-4, the detail level of the 'TREE_IO' package has been increased to show subprograms. The detail level of other packages has been reduced so that they appear only as points.

Other 'secondary views' (eg the graph view) can be used to provide supplementary information or present information in a more meaningful way.

A.6 Script Mode

SEE-Ada supports task facilitation by providing a Script Mode Interface. This interface can support the setup of the environment. It also supports the preparation of custom descriptions for particular tasks.

Figure A-5 provides an example of a script that can be used to support the evaluation of coding practices. The user can either step through each script action or 'Run' the script in which case the script will automatically execute each action until it reaches a 'stop' command. The script begins by opening the system to be evaluated and then automatically displays the Subsystem View called 'CSCI_DESIGN'. The user then interacts with the environment to select which sections of the system will be evaluated. A Layers View showing the Ada Compilation units for this section of the system is then displayed. The view is then tailored to provide a compact representation onto which other information can be superimposed. The script then supports various evaluation activities. The first phase of the evaluation (at Step 3) is to check for usage of anonymous types. The script selects an attribute that will indicate the use of anonymous types and overlays this information on the Layers View. The user gets an immediate visual indication of whether the code complies with this criterion. The user can then interact with the environment to conduct a further analysis. For example, the user may wish to see how the feature is used in the actual source code or may wish to change the colour mappings to highlight units with the highest proportion of non-compliances.

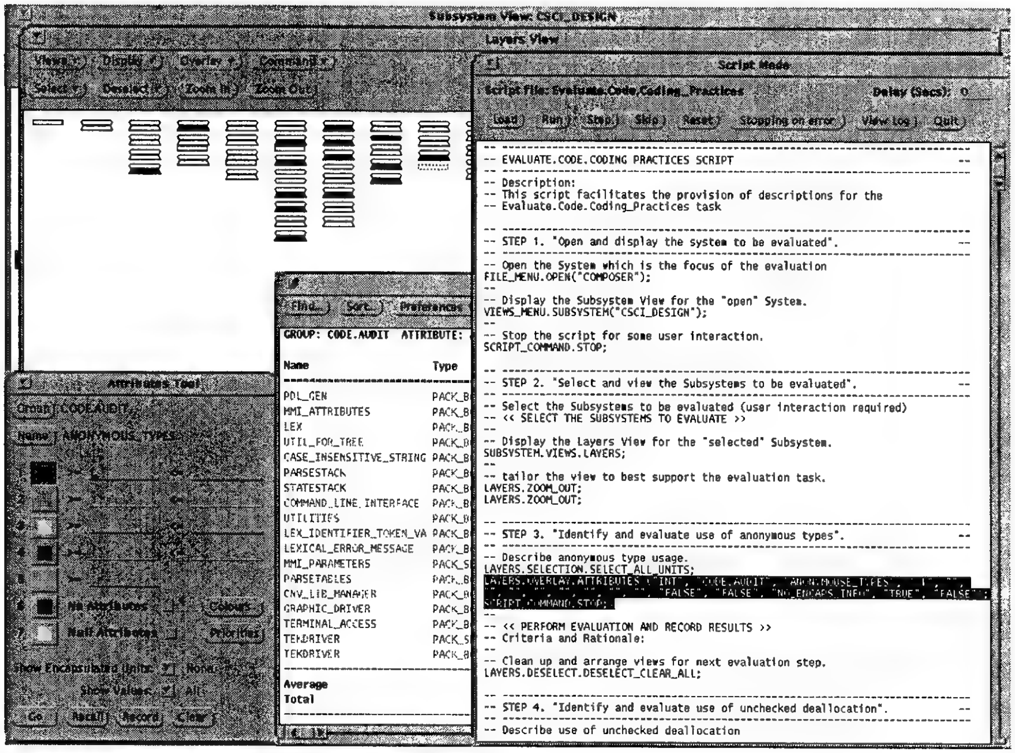


Figure A-5. SEE-Ada Script Mode

The user then moves on to the next stage of the evaluation by running or stepping through the script. The script automatically 'cleans up' the views and then produces a visual description which will support the next stage of the evaluation.
By using scripts in this manner, the set of actions that need to be undertaken for these types of evaluation tasks can be recorded and enacted. Relevant descriptions are provided to support the analyst. Although custom descriptions are produced, these can be adapted by users to help support their specific information needs.

Appendix B - Task Definitions and Examples

This appendix lists task definitions that were developed for the FCS project. The selections that could be made from the SEE-Ada usage monitor are provided as examples of instances of the various classes of tasks.

I. **Evaluation:** Systematic determination of the extent to which an entity meets its specified criteria.

- Evaluate.Code
- Evaluate.Code.Ada_95_Compatibility
- Evaluate.Code.Architecture
- Evaluate.Code.Coding_Practices
- Evaluate.Code.Commenting
- Evaluate.Code.Error_Handling
- Evaluate.Code.Init_and_Shutdown
- Evaluate.Code.Layout
- Evaluate.Code.Multiprocessing
- Evaluate.Code.Naming
- Evaluate.Design

II. **Analysis:** Examination for the purpose of understanding. (eg may need to perform analysis to gain sufficient understanding to specify a solution to a problem).

- Analyse.Code
- Analyse.Code.Characteristics
- Analyse.Code.Dependencies
- Analyse.Code.Features
- Analyse.Design
- Analyse.Design.Dependencies
- Analyse.Design.Features
- Analyse.Product
- Analyse.Product.Problems

III. **Inspection:** Examination to identify potential risks and product problems based on user's knowledge and experience. Inspections differ from evaluations in that they do not use specified criteria.

- Inspect.Code

- Inspect.Code.Configuration
- Inspect.Code.Descriptiveness
- Inspect.Code.Maintainability
- Inspect.Code.Numerical_Methods
- Inspect.Code.Portability
- Inspect.Code.Reliability
- Inspect.Code.Requirements_Allocation
- Inspect.Code.Safety
- Inspect.Design
- Inspect.Design.Requirements_Allocation
- Inspect.Design.Structure

IV. Monitoring: Examination to assess progress and status of activities.

- Monitor.Configuration_Status
- Monitor.Integration_Status
- Monitor.Release_Status

V. Recording: Capture or update of persistent information to support future needs.

- Record.Code
- Record.Code.Features
- Record.Code.Requirements_Allocation
- Record.Design
- Record.Design.Changes
- Record.Design.Features
- Record.Design.Requirements_Allocation
- Record.Evaluation_Actions
- Record.Inspection_Actions

VI. Reporting: Preparation of persistent information to support the needs of a requestor.

- Report.Evaluation_Results
- Report.Inspection_Results
- Report.Product_Characteristics
- Report.Product_Features

- Report.Product_Structure
- Report.Requirements_Allocation

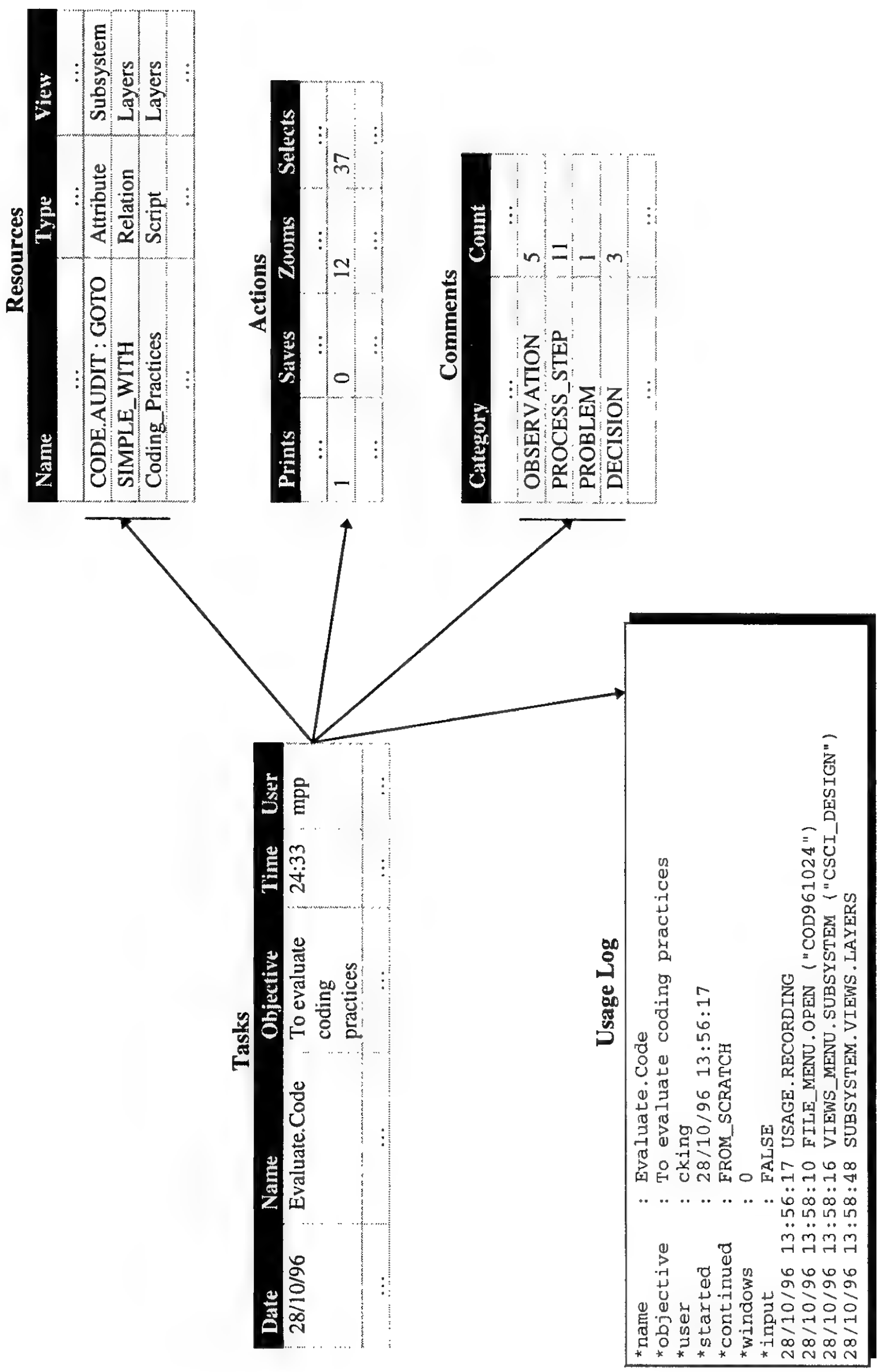
VII. Demonstration: Interactive presentation of information (eg to customers, management, auditors etc).

- Demonstrate.Config_Mgt
- Demonstrate.Config_Mgt.Status
- Demonstrate.Evaluation
- Demonstrate.Evaluation.Actions
- Demonstrate.Evaluation.Status
- Demonstrate.Integration
- Demonstrate.Integration.Status
- Demonstrate.SEE_Ada
- Demonstrate.SEE_Ada.Concepts
- Demonstrate.SEE_Ada.Usage
- Demonstrate.Test
- Demonstrate.Test.Coverage
- Demonstrate.Test.Results

VIII. Familiarisation: Observations to help gain general understanding of products, resources, or processes.

- Familiarisation.Product
- Familiarisation.Product.Attributes
- Familiarisation.Product.Features
- Familiarisation.Product.Structure
- Familiarisation.SEE_Ada
- Familiarisation.SEE_Ada.Concepts
- Familiarisation.SEE_Ada.Features
- Familiarisation.SEE_Ada.Usage

Appendix C - UMAS Data Schema



Appendix D - Usage Profiles

This appendix contains the subset of task profiles from the FCS project referred to in §5.3.

Against each task definition is listed the person who recorded the task, the time the task took to execute, the number of different SEE-Ada views used, the number of attributes applied, the number of relations displayed, the total number of user-initiated commands, the total number of comments made by the user, the number of print commands, the number of scripts used and the total number of script-initiated commands.

Task ID	Task Definition	User	Time	Views	Attributes	Relations	User Actions	Comments	Prints	Scripts	Script Actions
1	Monitor	cking	00:27:35	4	0	4	41	5	2	0	0
2	Monitor.Configuration_Status	cking	00:50:08	3	16	0	28	2	2	0	0
3	Monitor.Configuration_Status	cking	00:27:40	2	3	3	0	0	0	0	0
4	Inspect.design.requirements_allocation	rdolan	02:14:52	1	13	0	37	7	2	0	0
5	Inspect.Code	cking	00:21:00	3	19	4	0	1	0	0	0
6	Inspect.Design.Structure	cking	00:45:09	3	6	0	32	2	1	0	0
7	Inspect.Code	cking	00:34:53	3	15	0	38	3	1	0	0
8	Inspect.Code	cking	00:13:23	3	0	0	10	2	1	0	0
9	Inspect.Code.Configuration	jparsons	00:48:03	4	0	0	0	0	0	0	0
10	Inspect.Code	rdolan	00:26:38	3	0	0	11	0	1	0	0
11	Inspect.Code	cking	00:26:51	3	6	0	0	0	0	1	6
12	Record.Design.Changes	cking	00:22:33	2	0	0	15	0	0	0	0
13	Analyse.Design	rdolan	00:22:04	2	0	0	3	1	1	0	0
14	Evaluate.Code.Coding_Practices	cking	00:18:53	3	2	0	0	2	0	1	4
15	Evaluate.Code	cking	01:35:59	3	23	0	27	6	1	1	17
16	Evaluate.Code.Coding_Practices	cking	01:49:48	4	30	2	0	37	0	1	24

DISTRIBUTION LIST

Capturing and Analysing Usage of Interactive Computer-Based Tools
(DSTO-RR-0119)

M.P. Phillips, R.J. Vernik

	Number of Copies
AUSTRALIA	
DEFENCE ORGANISATION	
Task sponsor:	
FASDM	1
S&T Program	
Chief Defence Scientist)	
FAS Science Policy)	1 shared copy
AS Science Corporate Management)	
Director General Science Policy Development	1
Counsellor, Defence Science, London	Doc Control sheet
Counsellor, Defence Science, Washington	Doc Control sheet
Scientific Adviser to MRDC Thailand	Doc Control sheet
Director General Scientific Advisers and Trials)	1 shared copy
Scientific Adviser - Policy and Command)	
Navy Scientific Adviser	1 copy of Doc Control sheet and 1 distribution list
Scientific Adviser - Army	Doc Control sheet and 1 distribution list
Air Force Scientific Adviser	1
Director Trials	1
Aeronautical & Maritime Research Laboratory	
Director	1
Electronics and Surveillance Research Laboratory	
Director	1
Chief Information Technology Division	1
Research Leader Command & Control and Intelligence Systems	1
Research Leader Military Computing Systems	1
Research Leader Command, Control and Communications	1
Executive Officer, Information Technology Division	Doc Control sheet
Head, Information Architectures Group	Doc Control sheet
Head, C3I Systems Engineering Group	Doc Control sheet
Head, Information Warfare Studies Group	Doc Control sheet
Head, Software Engineering Group	Doc Control sheet
Head, Trusted Computer Systems Group	Doc Control sheet
Head, Advanced Computer Capabilities Group	Doc Control sheet

Head, Computer Systems Architecture Group	Doc Control sheet
Head, Systems Simulation and Assessment Group	Doc Control sheet
Head, Intelligence Systems Group	Doc Control sheet
Head, CCIS Interoperbility Lab	Doc Control sheet
Head Command Support Systems Group	Doc Control sheet
Head, C3I Operational Analysis Group	Doc Control sheet
Head Information Management and Fusion Group	Doc Control sheet
Head, Human Systems Integration Group	Doc Control sheet
Task Manager	1
Author	1
Publications and Publicity Officer, ITD	1

DSTO Library and Archives

Library Fishermens Bend	1
Library Maribyrnong	1
Library DSTOS	2
Australian Archives	1
Library, MOD, Pyrmont	Doc Control sheet

Forces Executive

Director General Maritime Development,	Doc Control sheet
Director General Land Development,	Doc Control sheet
Director General C3I Development	1

Navy

SO (Science), Director of Naval Warfare, Maritime Headquarters Annex, Garden Island, NSW 2000.	Doc Control sheet

Army

ABCA Office, G-1-34, Russell Offices, Canberra	4

Intelligence Program

DGSTA, Defence Intelligence Organisation	1

Corporate Support Program (libraries)

TRS Defence Regional Library, Canberra	1
Officer in Charge, Document Exchange Centre (DEC),	1
US Defence Technical Information Center,	2
UK Defence Research Information Centre,	2
Canada Defence Scientific Information Service,	1
NZ Defence Information Centre,	1
National Library of Australia,	1

Universities and Colleges

Australian Defence Force Academy Library	1
Head of Aerospace and Mechanical Engineering	1
Deakin University, Serials Section (mlist), Deakin University Library, Geelong, 3127	1
Senior Librarian, Hargrave Library, Monash University	1
Librarian, Flinders University	1

Other Organisations	
NASA (Canberra)	1
AGPS	1
State Library of South Australia	1
Parliamentary Library, South Australia	1

OUTSIDE AUSTRALIA

Abstracting and Information Organisations	
INSPEC: Acquisitions Section Institution of Electrical Engineers	1
Library, Chemical Abstracts Reference Service	1
Engineering Societies Library, US	1
Materials Information, Cambridge Scientific Abstracts	1
Documents Librarian, The Center for Research Libraries, US	1
Information Exchange Agreement Partners	
Acquisitions Unit, Science Reference and Information Service, UK	1
Library - Exchange Desk, National Institute of Standards and Technology, US	1
SPARES	10
Total number of copies:	**60**

Page classification: UNCLASSIFIED

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA				1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)	
2. TITLE Capturing and Analysing Usage of Interactive Computer-Based Tools			3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION) Document (U) Title (U) Abstract (U)		
4. AUTHOR(S) M.P.Phillips and R.J.Vernik			5. CORPORATE AUTHOR Electronics and Surveillance Research Laboratory PO Box 1500 Salisbury SA 5108		
6a. DSTO NUMBER DSTO-RR-0119	6b. AR NUMBER AR-010-406		6c. TYPE OF REPORT Technical Report	7. DOCUMENT DATE December 1997	
8. FILE NUMBER N9505/13/52	9. TASK NUMBER 94/081	10. TASK SPONSOR FASDM	11. NO. OF PAGES 43		12. NO. OF REFERENCES 12
13. DOWNGRADING/DELIMITING INSTRUCTIONS N/A			14. RELEASE AUTHORITY Chief, Information Technology Division		

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE CENTRE, DIS NETWORK OFFICE, DEPT OF DEFENCE, CAMPBELL PARK OFFICES, CANBERRA ACT 2600

16. DELIBERATE ANNOUNCEMENT

No limitations

17. CASUAL ANNOUNCEMENT Yes

18. DEFTEST DESCRIPTORS

Software Engineering
Computer-aided Software Engineering
Human Computer Interface

19. ABSTRACT
This report argues that computer-based tools should incorporate features that support the capture and analysis of usage information, particularly if these tools are to be used as part of a research program. In addition to discussing the issues that need to be considered in terms of tool instrumentation and analysis support, this report provides details of experiences gained through the instrumentation of a Computer Aided Software Engineering (CASE) tool.

Page classification: UNCLASSIFIED